REVEAL®

A Keystone of Modern Systems Engineering

Reference: S.P0544.19.1
Issue: 1.2
Status: Definitive
24th July 2001
## Contents

1. Management Summary  
2. Introduction  
3. Background to the Approach  
4. Principles and Definitions  
5. The REVEAL Process  
6. Defining the Problem Context  
7. Identifying Stakeholders and Eliciting Requirements  
8. Analysing and Writing Requirements  
9. Verification and Validation  
10. Conflict Management  
11. Maintenance and Management  
12. Conclusions  

Document Control and References  
3  
4  
5  
9  
17  
19  
23  
26  
31  
35  
39  
45  
46
1 Management Summary

The world of systems engineering has known for some time just how difficult it is to define and maintain requirements on large or complicated projects, and how costly failures frequently stem from the requirements elicitation stage. For years engineers and business analysts have known the importance of accurate and adequate requirements, but have struggled to understand how to describe the environment, elicit users’ real needs and turn them into useful specifications of effective systems.

In our experience there are three fundamental problems of Requirements Engineering:

1. People often find it hard to articulate their requirements.
2. When requirements are identified, they will often conflict.
3. Even if requirement conflicts are resolved, the requirements will change.

As a response to these problems, Praxis Critical Systems has developed a method we call REVEAL. This combines recent research and the latest techniques into a well structured, practical, and clear method for eliciting and defining requirements; for improving specification clarity; and then for managing change to those requirements.

REVEAL is a true engineering method since it is the systematic application of scientific principles. The scientific principles have been published by Michael Jackson, and are augmented with practical engineering experience to provide a rigorous process.

REVEAL accepts the problems as a starting point in the RE process:

- It recognises that requirements are about the real world and uses a systematic elicitation process to identify and document what the users need as well as the environment within which the system must operate.
- It recognises that, because many people have an interest in the provision of the system, requirements will conflict. The REVEAL process actively identifies conflicts and provides guidelines for analysing and resolving conflicts to derive a conflict-free specification.
- It acknowledges that requirements will change and incorporates change management principles that can future-proof requirements against foreseeable change. REVEAL also provides a process for managing unforeseen changes.

Praxis Critical Systems has been applying the principles underlying REVEAL for over ten years in demanding and sometimes critical environments, using a variety of formal methods, object-oriented and natural language approaches and with a variety of tools. We have used REVEAL successfully on many projects including business, safety and security critical systems and have extensive experience of managing and reducing these risks. REVEAL helps eradicate costly errors.

---

1 REVEAL is a registered trademark of Praxis Critical Systems Limited.
2 Introduction

2.1 Background
Praxis Critical Systems is a company based in the UK specialising in the engineering of critical software-intensive systems. Praxis Critical Systems provides a variety of Requirements Engineering (RE) services, both in the development of custom systems, and in the provision of RE functions for clients involved in larger developments.

REVEAL is Praxis Critical Systems’ proprietary method for RE. It captures many of the fundamental principles appropriate for successful Systems Engineering which Praxis has been applying for over 10 years. REVEAL is a method that has been developed over the last five years and is routinely used on projects within the finance, rail, and aerospace markets.

2.2 Purpose
The purpose of this white paper is to provide a detailed technical description of REVEAL.

2.3 Scope
The white paper is a technical summary of REVEAL covering the major principles and the key process stages. It does not replace the existing training material and does not contain much of the detailed guidance, which is documented within the REVEAL training course.

2.4 Structure
Section 3 provides the background to the REVEAL method. Section 4 identifies the principles upon which REVEAL is based. These principles provide clear definitions of the key terms: Requirements, Specifications, and Domain Knowledge. Section 5 introduces the REVEAL process and sets the scene for the detailed description of each of the major process stages contained within Sections 6-11. The specific objectives of each stage are identified, the principles that are relevant to the objectives are described, and the process steps then identified. Key issues relevant to the stage are then discussed and, where appropriate, observations on candidate techniques that implement these principles are made. The intention of the process step descriptions is to share our experiences and insights on the undertaking of these stages. The combined execution of the stages forms the overall REVEAL method.

Finally Section 12 concludes the paper.
3 Background to the Approach

3.1 The Problems of Requirements

It has long been known that the existence of complete, correct, and usable requirements is a prerequisite for successful project delivery [1]. Problems remain and failings in RE are still cited [2] as one of the principal causes for over-run, over-budget, and under-achieving projects. In our experience there are three fundamental problems to the RE activity.

1 Expression: People often find it hard to articulate their requirements.
2 Conflict: When requirements are identified, they will often conflict.
3 Change: Even if requirements conflict is resolved, the requirements will change.

Consider an airport manager who has a need to increase the flow of passengers through an existing airport. The airport manager might articulate this need by describing the requirements of a solution such as a passenger ‘check-in’ desk that has a higher degree of automation than currently available. Such a solution may or may not meet the airport manager’s actual requirement which might be better served by increasing the throughput of aircraft by way of a more sophisticated air traffic control system. A true understanding of the airport manager’s problem can only be gained, and thus the most appropriate solution properly identified, through an understanding of the airport manager’s world (or domain). Only when the domain is understood can the requirements be effectively identified and the expression problem avoided.

Having correctly identified and understood the airport manager’s requirements, conflicts may be identified. For example, a consequence of increasing aircraft throughput at the airport is that the local inhabitants must suffer increased noise pollution – there is a conflict between the airport manager and the local residents. This problem of conflict is a consequence of constructing systems that must satisfy the needs of a variety of different individuals. Complex systems typically must meet the needs of political wishes, legal and social conditions, user requirements, etc. and these needs will not necessarily be in agreement. For the airport example, the conflict might be resolved by increasing aircraft throughput during the day (satisfying the airport manager), but decreasing aircraft throughput during the night (satisfying the local inhabitants).

Once a requirements conflict is resolved it need not stay resolved for the lifetime of a system. Changing requirements reflects the changing nature of the world and is an inevitable project occurrence. The national air safety regulations for our airport may change, providing a fundamental restriction on the number of aircraft permitted to circle the airport and thus the number of passengers who may land in any given hour. If such a change were to occur, the change would have to be evaluated and the airport manager’s requirements reconsidered.

Praxis do not believe that these three problems can be avoided – however we do believe that their effects can be ameliorated through successful management. The
problems are ‘facts of life’ for projects and if a RE method does not address these problems then it is our belief and experience that the method will be unsuccessful in its application – the method will not be industrial strength. REVEAL is a RE method that was developed to address these problems (see Section 3.4).

The purpose of this paper is to describe REVEAL, but also to reflect on those aspects of RE technology that have emerged to date, to comment on those features we have found useful in practice, and to identify those features we have found unhelpful. In order to set the context for the main content of this paper, it is useful first to consider the answers to two questions.

1. Who actually needs requirements?
2. What should the RE process achieve?

The purpose of posing these questions is to enable the reader to understand the context within which REVEAL resides, and in particular to best understand how REVEAL helps to mitigate the occurrence of the expression, conflict and change problems.

3.2 Who Needs Requirements?

In a general model of development suppliers construct systems for customers. The term “customers” is used rather loosely here to mean those people who want some problem solved, and “suppliers” is used to mean the people and organisation that will provide a solution. In Section 7, the term stakeholder is used to describe any party who has an interest in a project’s development – customers and suppliers are both types of stakeholders. Figure 1 indicates the role that the RE process has in bridging the gap between suppliers and customers, together with some incomplete examples of stakeholders of each type for illustration.

![Figure 1 – The Customer/Supplier Relationship](image)

Users are those stakeholders who will directly use the system. Their needs must clearly be considered in defining a system, for example the ‘check in’ clerks in our previous airport example. Other people affected are those who do not use the system directly, but will be influenced by it (for example the aircraft passengers). The purchaser is the party who pays for the system – typically purchasers will have very different requirements from users, they will be more interested in cost than convenience, or may even wish to eliminate users altogether. The information obtained from all of these customers provides a significant input to understanding what it is that a system should achieve – the customer will judge the success of the system against these requirements. However, the requirements are also necessary for
the suppliers. The requirements assist the developers in understanding what must be constructed, they assist the verification and validation (V&V) staff in understanding how the suitability of the system can be demonstrated, and they assist the management in understanding how progress may be monitored.

The RE process therefore provides a bridge between the customers and the suppliers. However the nature of the requirements process that characterises this bridge, and the actual stakeholders involved, will depend upon the context of the development. Consider three different kinds of development.

**Custom Development.** Most work in RE has focussed on custom development, where there is a single customer and a single project. Custom development is in some ways the most straightforward case. Most software developments used to be custom developments, although hardware developments were typically products.

**Product Development.** Washing machines, videos, and cars have almost always been products, not custom developments. It is becoming increasingly common for software to be developed as a product. The requirements process is then quite different to a custom development, because there is no one customer. Usually some organisation such as a marketing department has to act as a surrogate for the customer. In product development, requirements are led more by what is possible than what is needed. Nevertheless it is important to be clear about what needs a particular product will or should address, and what are outside its scope.

**Competitive Procurement.** In competitive procurement, the requirements definition provides the baseline against which competitive companies will be assessed. If the definition is not clear and complete enough, suppliers will be evaluated on the basis of a misunderstanding of what they have to produce, and it is unlikely that they will produce what is really wanted, especially if the contract is fixed price.

The focus of the RE process will differ depending upon the nature of the customer/supplier development. We have used REVEAL in a variety of commercial arrangements, including Rapid Application Development (RAD). Clearly the relevance of some of the REVEAL process stages changes depending upon the development approach. The full REVEAL method provides guidance on tailoring REVEAL depending upon the commercial context.

### 3.3 What should the Requirements Process Achieve?

The purpose of the RE process is not simply to arrive at a complete, correct, and usable description of requirements. The RE process is important in its own right – that is, if the RE process has been successful there should be recognisable material benefit for the stakeholders who have taken part in the RE process. The process is one of learning, negotiation and of building and developing trust, and it is a process that is shared by all parties. Participation in the process can be just as important for those concerned as the production of the end result – involvement in negotiations.
make stakeholders aware of the history and reasoning behind any compromises, which helps “buy-in” to the solution from an early stage.

An important outcome of negotiation is the resolution of conflicts between functionality, cost and development timescales. This is frequently called Value Engineering, which is the development of artefacts that are perceived as good value: it means not necessarily satisfying every functional requirement regardless of cost, but assessing the value of each aspect of the system and incorporating those that are cost-effective. The development of trust and of understanding obtained during the RE process, improves the likelihood that systems will be developed that add value to the customer, and do not simply meet the perceived and documented requirements.

3.4 What is REVEAL?

REVEAL is a systematic principled method for the elicitation, specification, and management of systems requirements. REVEAL is based on the scientific principles published by Michael Jackson [3, 4], augmented with practical engineering guidance derived through Praxis Critical Systems’ experience. Specifically REVEAL has encoded Jackson’s principles into a complete, systematic, and proven RE method.

REVEAL provides principled guidance on the RE activity from the early phases of the establishment of problem context, through to the identification of stakeholders, the elicitation and recording of requirements, their verification and validation, and on to their use and maintenance. REVEAL includes a framework and a process that does not enforce particular notations or tools. Praxis Critical Systems has experience of applying the REVEAL method with a variety of formal methods, object-oriented approaches, and natural language – as well as a variety of tools.
4 Principles and Definitions

4.1 Introduction

Clearly defined RE principles and definitions are a pre-requisite for a systematic RE method. Jackson introduces the generic “World and the Machine” model [3] as a basis for a number of RE principles. This model is adopted within REVEAL.

4.2 Principles

Machines are built in order to bring about some improvement to the World. In the airport example discussed in Section 3.1, the World contains passengers, aircraft, check-in desks, check-in staff, etc. It is this real World that we wish to influence by the construction of some new Machine, or System. In our terms, a Machine can be any socio-technical artefact that is in some way to be constructed. The new Machine could be a new check-in system, or a revised passenger policy for the airlines. The former includes perhaps a single piece of technology, whereas the latter may include the construction and synchronisation of a vast range of equipment, modification of legislation, changes of behaviour in the population, etc. Those aspects of the real World that are relevant to the problem are termed the Application Domain. Whenever a Machine (M) is introduced to the World (W), it interacts with the World through a shared interface (see Figure 2).

![Figure 2 – The World and the Machine](image)

If our airport were to have a new Air Traffic Control System introduced into it, then the Machine might include Air Traffic Control system terminals, a radar system, and operators. Items that might occur in the interface between the World and the Machine (Interface in Figure 2) are those phenomena that are shared between the World and the Machine – these might include transponders, voice communications, radar aerials, etc.

Phenomena in the interface are shared because they are in some sense visible to both the World and the Machine. For example, an operator may transmit some voice message to an airborne aircraft. The voice message is shared between the Machine and the World. The Machine creates the voice message, but the World (in this case the aircraft’s pilot) pays attention to it. It is through this shared phenomenon that the Machine seeks to influence the behaviour of the World. Other shared phenomena will be controlled by the World and responded to by the Machine. All shared phenomena are either controlled by the Machine, or controlled by the World. The identification of the boundary, between the Machine and the World, and the nature of the shared
phenomena is one of the most difficult parts of RE. This identification is described in Section 6.

The “World and the Machine” model provides the scientific basis for a number of definitions.

4.3 Definitions and Examples

The following definitions are based on those provided by Jackson [4].

Requirements (R) are statements about things in the World (W) that we want the Machine to help make true.

A System Specification (S) describes a Machine’s (M) external behaviour.

Specifications include only shared phenomena in the interface between the World and the Machine.

Specifications can only constrain shared phenomena that the Machine can control.

A Design is a statement about the Machine itself. It describes things in M.

The phrases ‘User Requirements’ or “User Needs” are increasingly being used to describe what in REVEAL would be called ‘Requirements’, and ‘System Requirements’ is being used to describe what in REVEAL would be called ‘Specification’. We prefer to adopt Jackson’s definitions as a basis. There are a number of observations that should be made on these definitions.

Some Requirements statements may also be Specifications. Requirement statements are also specifications if they define constraints on shared phenomena that are controlled by the Machine – for example a Requirement statement requiring that a Machine interface with some existing legacy equipment.

Requirements are defined in terms of ‘…things in the World…’. That is Requirements are not concerned with a description of the Machine. Requirements are also concerned with describing things that ‘…we want the Machine to help make true’. The implication is that it is not solely the Machine that brings about the Requirements, other properties (independent of the Machine) may be required to bring about the Requirements. This issue, and the underlying definitions, is best illustrated with an example.

Consider the problem of a pedestrian wanting to cross a road. An appropriate high level requirement for this problem might be as follows.

\[ R1: \text{Pedestrians shall be allowed safely to cross roads that contain traffic.} \]

Note that the statement is made entirely in terms of entities within the World and does not mention the Machine at all. From Figure 3 it can be assumed that the World includes people, roads, and cars for example.
It is perhaps the case that the requirement $R_1$ is intended to be brought about by the construction of a set of traffic lights together with some digital traffic light controller. Of course it may equally be valid to meet this requirement by changing traffic legislation in the region, by building a bridge, or perhaps by constructing a subway. However in this case, the requirement is to be met by the construction of traffic lights.

We note that it is very common to have some high level understanding of the proposed Machine solution (e.g. the construction of a set of traffic lights) even at this stage. It is possible to use the REVEAL concepts at this stage without considering particular design options, for example in analysing strategic business requirements. However without some notion of the type of solution (e.g. the modification of traffic policy or the construction of traffic lights) the requirements cannot be expressed in any more detail and no progress towards a solution will be made. Note that if design concepts are being evaluated, a number of solutions may be considered, but ultimately one must be chosen before detailed progress can be made. We return to this issue in Section 4.5.

The Specification of the traffic lights might include statements such as the following.

$S_1$: When the buttons are pressed, cars shall be presented with red lights and pedestrians shall be presented with green lights.

Note that the statement describes entities that are shared phenomena between the World and the Machine (i.e. buttons and lights). Can a Machine, which is built and installed to satisfy statements such as $S_1$, by itself achieve the satisfaction of statements such as $R_1$? Clearly the answer is that statements such as $S_1$ are necessary but not sufficient because there are a number of assumptions which must hold true to ensure that $R_1$ is satisfied. For example:

$D_1$: Drivers stop at red lights.

$D_2$: Pedestrians push the button when they want to cross.

$D_3$: Pedestrians cross when they have a green light.

Jackson points out that these types of statements are properties of the World that must also hold if the introduction of the Machine is to bring about the satisfaction of the requirements. Jackson call these properties of the World ‘Domain Knowledge’ ($D$). Note that Domain Knowledge is properties of the World that we know to be true, and
Requirements are properties of the World that we wish to make true (caused in part by the introduction of the Machine).

This recognition of the role of Domain Knowledge in the RE process leads Jackson to suggest the following relationship, which must hold if a Machine is to be introduced to the World to bring about some requirements.

$$D, S \vdash R$$

The Satisfaction Argument should be read as follows: using the relevant properties of the World ($D$), when combined with the specification of the behaviour of the Machine to be constructed ($S$), it is possible to show ($\vdash$) that the Requirements ($R$) will hold. The Satisfaction Argument therefore means that the specification of the Machine is correct if, and only if, we can show that the specified behaviour of the Machine and the properties of the domain into which the Machine is put, are together sufficient to achieve the requirements. Within REVEAL the Satisfaction Argument provides a framework for the whole RE process and for the overall RE documentation.

4.4 The Use of Language - Designations

The overall structure of the requirements documentation can be derived from the Satisfaction Argument, but clear requirements descriptions also require the clear use of language. One of the key difficulties in using clear language is the confusion that arises through the use of general terms. Jackson for example [4] discusses the use of the term “phone call”. When we use the term phone call does it include just the period of conversation, or does it include making and terminating the communication – what about a conference call? In order to address this problem of description Jackson proposes [4] distinguishing between ‘designations’ and ‘definitions’.

**Designations** are ground terms that define a basic vocabulary.

**Definitions** are then composites of ground terms that then allow more sophisticated concepts to be constructed.

For example the definition of “phone call” can be constructed from a combination of designations that define phrases like “off hook”, “dialed digit”, etc.

The use of these basic principles can have a dramatic effect on the readability of requirements descriptions. Specifically the use of precise and unambiguous language increases the likelihood of producing refutable descriptions – that is, descriptions that can be shown to be correct or incorrect. For example the statement that “All cars can come to a stop from 60mph within 2 seconds” is refutable – one can simply find a car for which this claim does not hold to show it is incorrect. If no such cars can be found then the statement is correct. However, the statement that “All cars can stop quickly” is not refutable, as the term “quickly” is imprecise – it means different things to different people and is therefore ambiguous.

These principles and definitions underpin the REVEAL process. They provide a means for labelling particular types of statements and a framework for relating properties of the World to properties of the Machine. All three types of information ($D$, $S$, and $R$) are important, and the type of any particular statement must be stated.
This labelling activity requires care, as a statement’s type is not always clear in isolation. Consider for example the statement that “aircraft are always separated by 7000 metres horizontally and 400 metres vertically” – is this Domain Knowledge or Requirements? The answer is that is depends on the intent of the statement.

4.5 The Use of the Principles to Structure the RE Activity

The definitions identified in Section 4.3 are used differently depending upon the context of the development. Specifically there are some important differences between carrying out RE activities in a single system procurement when compared to the procuring of multiple systems (where the integrated collection of systems must satisfy the requirements). We shall consider first the problem of single system procurement, and the application of the principles. We shall consider the task of moving from requirements to design, the role of non-functional requirements, and the obligations that the various classes of stakeholder have in the provision of information. The remainder of the paper does not explicitly distinguish between the different types of procurement.

4.5.1 Application within a Single System Procurement Environment

Figure 4 captures the relationship between the types of information recorded, and the level of detail recorded of that type of information.

![Figure 4 – Information Management within a Single System Procurement](image)

The bottom left of Figure 4 denotes the initial statement of the overall requirements (Rgen), for example “improve air traffic flow over oceanic airspace”. In order to generate a system implementation from Rgen, we must consider two dimensions.

- It is necessary to move from an understanding of what is required in the World (the general requirements Rgen and detailed requirements Rdetailed) to a description of the Machine that is to be produced (the high level design HLD and detailed design Design).
- It is necessary to increase the amount of detail understood to produce the detailed requirements (Rdetailed) from the general requirements (Rgen), and to produce the detailed design (Design) from the high level design (HLD).
Notice that these are different dimensions. Two routes for implementing a Machine from the general requirements are shown on Figure 4, but neither of them is realistic.

Route 1 proceeds directly from the general statement of requirements to a high level design of the Machine and from there to the detailed design. The problem is that the general statement of requirements gives little clue about how to generate a high level design. Indeed, it may give no hint at all as to what sort of Machine might be used to meet the requirements: should it be a new Air Traffic Control system, or new legislation which encourages airlines to fill their planes more efficiently? All the design decisions in going from \( \text{Rgen} \) to \( \text{HLD} \) are therefore suspect. A high level design describes the structure of the Machine but does not say what it will do. The detailed design cannot be derived without further understanding of the requirements.

Route 2 proceeds from the general statement of requirements to a detailed statement of the requirements before the corresponding detailed design is then produced. However this route is also problematic, because to add detail to the requirements it is necessary to have some idea about what sort of Machine is required – is it an Air Traffic Control system at all? Maybe we would be better off making aircraft hold more passengers? Until we have some idea what sort of Machine we are trying to build, we cannot say what its detailed requirements are. Only when we have decided that we are going to build an Air Traffic Control system can we start specifying requirements for aircraft separation, controller communications and so on.

Therefore Requirements and Specifications have to be developed concurrently. So the most practical route is the one shown in Figure 5. Starting from the general requirements such as “improve traffic flows”, we refine these into more specific requirements such as “reduce separations”, at the same time specifying what the Air Traffic Control system needs to do to reduce separations. We thus arrive at a collection of detailed requirements that include a system specification. That system specification can be given to a supplier to design and produce the Machine itself.

![Figure 5 - Producing Detailed Designs from General Requirements](image_url)

The process of getting from general to detailed requirements has some similarity with getting from Requirements to Specifications. It may require Domain Knowledge to show that \( \text{D}, \text{Rdetailed} \vdash \text{Rgen} \) just as domain knowledge is required to demonstrate the Satisfaction Argument. For example we need to understand how reducing...
 separations will improve traffic flows. The difference is in the nature of the result. Detailed requirements are still requirements: things that we want to be true in the World (smaller aircraft flight separations cannot be brought about purely by an Air Traffic Control system for example).

The high-level design can often be produced in parallel with the detailed requirements and functional specifications through consideration of the non-functional requirements. Non-functional requirements are those requirements such as reliability, timeliness, etc that drive the overall structure of a system. As the high-level design typically follows from such non-functional requirements, the structure can often be produced largely independently of the function. Figure 5 shows this parallelism.

In a procurement context the “High Level Design”, shown in Figure 5, may be replaced by “Analysis of Candidate Solutions”. This activity involves considering possible solutions to the mission needs and evaluating their suitability.

4.5.2 Application within a Multiple System Procurement Environment

Many procured systems are created from an integration of smaller systems. We call the resulting integrated collection a ‘system of systems’. The application of the principles described above is more complex when procurement of a system of systems is required. Specifically, there is a need to consider what additional requirements are needed, and to consider how the requirements can be distributed between and across the constituent systems. Figure 6 provides an indication of the changed situation.

![Figure 6 – Multiple System Procurement](image)

When a single requirement is to be satisfied by more than one subsystem, the detailed requirements and specifications for each subsystem must be derived from the general requirement for the whole collection of systems. This is a design activity. One conceptual system (the outer ellipse of Figure 6) is being designed as a collection of smaller subsystems (the inner ellipses) plus a collection of interactions between them (the double-headed arrow). Distributing requirements in this way means that the organisation that has the general requirement also takes design responsibility for the collection of subsystems that will satisfy the requirement.

If each individual subsystem is the responsibility of a different supplier, then their requirements, and the design of their interactions, must be under the control of the acquiring organisation. This is because only the acquiring organisation has the
overall responsibility, knowledge and purchasing authority. No-one else is able to
determine whether a particular combination of subsystems is adequate, and to
negotiate the boundaries between all the subsystems to obtain an efficient and cost-
effective distribution of requirements. The term “requirements allocation” is often
used for this process, but it is a misnomer. The requirements on the individual
subsystems are derived from the general requirements, but they are not the same as
the general requirements.

Consider a communications system, which allows pilots to talk to air traffic
controllers, that consists of the following three subsystems: an airborne transceiver in
the aircraft; ground stations; a router for exchanging data between ground stations and
air traffic controller terminals. It is only the behaviour of all these subsystems
combined that can satisfy the requirement: “The system shall connect a pilot to the
correct air traffic controller”. Removing any of the three constituent subsystems will
result in the requirement no longer being met.

So the requirement cannot be allocated to any of the subsystems. Instead, each
constituent subsystem has derived requirements, such as being able to transmit or
receive on a certain set of frequencies.

We have argued that moving from overall requirements, to requirements on individual
subsystems is a design activity. This design therefore needs to be documented in
addition to the two levels of requirements. Such a design should at least include:

- a definition of what the constituent subsystems are;
- an explanation of how the subsystems work together to meet the overall
  requirements.

Remember that the detailed requirements may be about interfaces between constituent
subsystems. It may be possible to distribute these requirements between the
documents for the individual subsystems. However, in practice it is usually more
convenient to factor out such requirements into separate interface definition
documents. This factoring has the advantage of avoiding possible duplication
between the requirements documents of the two subsystems involved in the interface,
and of acting as a clear contract between the two systems. Similar arguments may
also apply for interfaces between a subsystem and a real-world entity that is outside
the complete system boundary (e.g. a user interface definition).
The REVEAL Process

Figure 7 – The REVEAL Process Model

The REVEAL process model (Figure 7) is derived from the principles contained within Section 4, and in particular is structured with a view to capturing the information required by the Satisfaction Argument. The process commences with understanding the context within which the Problem exists and exploring the bounds of the Problem. The Problem Context is the relevant part of the World i.e. those entities with which the problem-solving task is concerned. This first stage is called ‘Defining the Problem Context’ (Section 6) and includes understanding the boundary between the Machine and the World as well as obtaining an understanding of the relevant properties of the World. It is only by undertaking this stage that the scope of the problem can be assessed, and the appropriate vocabulary identified for the remainder of the RE process.

The second stage (‘Identifying Stakeholders and Eliciting Requirements’, Section 7), is concerned with the identification of the sources of requirements information (stakeholders) and the capturing, or elicitation, of information from them. The third stage (‘Analysing and Writing Requirements’, Section 8) is concerned both with the recording of information obtained from the stakeholders and with the derivation of a system specification from stakeholder requirements. The fourth stage (‘Verification and Validation’, Section 9) is concerned with confirming the accuracy of the information that has been elicited. These three stages can broadly be viewed as defining the information gathering, recording, and checking stages. These stages may be iterated during a single task (e.g. a stakeholder interview), as well as across the entire RE process (e.g. when requirements change, or a conflict is identified).

The first four stages are supported by an underlying process of resolving conflicts (‘Conflict Management’, Section 10). This stage exists to assist in the planned management of conflicting requirements that will emerge as information is gathered throughout the stages of the RE process.

After the requirements gathering, recording, and checking stages, the requirements information will be used throughout the remainder of the development life-cycle. REVEAL is not specifically concerned with these later stages of development such as
design. However, it is important that the principles REVEAL applies are maintained during the later stages of development and therefore it seems relevant to discuss some of the issues involved. Managing requirements under-pins the whole RE activity and for this reason it is shown as being applicable across the whole RE process – see the change problem in Section 3.1. Maintenance and management of requirements are described in ‘Maintenance and Management’, Section 11.

All stages support iteration. This is deliberate and is necessary to handle the conflict and change problems. The RE process can not be planned and scheduled as a series of activities without some appreciation of the underlying objectives. The RE process is complete when these underlying objectives are met, but is also re-commenced if any of the gathered information changes. The objectives of the RE process were identified in Section 3.3, but following the discussion in Section 4, these can be expressed as RE process completion criteria.

- **Stakeholder Completion Criterion**
  All the information contained within \( D \), \( S \), and \( R \) is agreed by all of the relevant stakeholders.

- **Adequacy Completion Criterion**
  The system specification and domain knowledge are shown adequate to satisfy the requirements (i.e. the Satisfaction Argument is met).

- **Stability Completion Criterion**
  The stability of information contained within \( D \), \( S \), and \( R \) is assessed and hence areas that are likely to change are identified. (This supports the management of requirements by helping to ensure that any changes are detected and their impact assessed.)
6 Defining the Problem Context

6.1 Introduction

The processes of RE are undertaken because there is a problem that needs to be solved. Problems exist within World. Requirements and problems cannot be articulated until the World itself has been understood. The Problem Context is the Application Domain, which is those entities in the World (people, systems, equipment, hardware, etc.) that are relevant to the problem, together with the boundary of the Machine, i.e. which part of the Application Domain forms the interface with the Machine.

Information regarding the Problem Context will be derived from existing knowledge of the application within a project, other stakeholders, documentation, etc. The purpose of this stage is to ensure that the requirements for the right problem will be developed. It is only by understanding the Application Domain that a valid set of Requirements can be found, and an appropriate Specification defined. It is the relationships defined by the Satisfaction Argument that provides the focus for this activity.

6.2 Process Steps

In order to define the problem context, a number of process activities are required. These are listed in Table 1.

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Capture a high level description of the problem.</td>
</tr>
<tr>
<td>2</td>
<td>Identify relevant parts of the Problem Domain.</td>
</tr>
<tr>
<td>3</td>
<td>Identify potential interactions with each other, and with the Machine.</td>
</tr>
<tr>
<td>4</td>
<td>Negotiate the boundary between the Application Domain and the Machine, consider what part of the Application Domain remains after the Machine has been introduced.</td>
</tr>
<tr>
<td>5</td>
<td>Start to define a vocabulary of terms that can be used to describe features of the Problem Context. This will be continued during the Analysis stage.</td>
</tr>
<tr>
<td>6</td>
<td>Start to describe the properties of the Application Domain, not just the parts that interact directly with the Machine. Use whatever notations and mechanisms are appropriate.</td>
</tr>
</tbody>
</table>

Table 1 – Process Steps for Defining the Problem Context
At the completion of this process, one would expect to be able to go on and begin to identify the stakeholders contained in the problem context who would act as a source for the requirements elicitation.

6.3 Discussion

Consider the World of Railway Transport. The Problem Context for railway transport might include trains, rail infrastructure, the train operating companies, the track, the signals, the network controllers, etc. Establishing the Context of Railway Transport provides the vocabulary by which ‘Railway Transport problems’ could be articulated. For example, the context would allow us to discuss scheduling problems involving the co-ordination of trains across the country, or problems about which tracks would be closed to the train operating companies at specific times.

The establishment of the Problem Context provides the vocabulary for identifying and describing the problem. This stage involves beginning to understand the scope of the problem, and also where the interface between the Application Domain and the Machine will reside. This activity will involve a determination of what the shared phenomena will be – which will be negotiated. This stage therefore clearly requires design knowledge in understanding what is and is not feasible to construct. We believe that consideration of the feasibility of design at this stage is absolutely crucial. Without this knowledge the identification of the boundary between the Application Domain and the Machine is not possible.

Understanding the nature of the boundary between the Application Domain and the Machine is rarely obvious. Consider a specific problem of railway signalling. The problem is concerned with providing railway network managers with location information for all the trains within their areas. One way in which train location information is provided to network managers is through the use of Track Circuits. Track Circuits provide information to a network manager when a passing train actuates them. In identifying the boundary between the Application Domain and the Machine, one of the most critical issues for this problem is to understand whether the Track Circuits are part of the Application Domain or part of the Machine. Figure 8 illustrates some of the entities that form part of this problem context.

![Figure 8 – The Track Circuit Boundary Problem](image-url)
If the Track Circuits are part of the Machine – that is, they may be changed or replaced as part of building the new Machine, perhaps replaced by more reliable technology – then it is reasonable to assume that the Machine will see trains move from one sequence of track to an adjacent sequence of track. If however the Track Circuits are part of the Application Domain, then the Machine can no longer assume that it will see sequential occupation of track sections, because track circuits may fail and not reliably indicate the positions of trains on the track. Different kinds of system specifications will be produced depending on whether the failure modes of the track circuit are taken account by the Machine.

In addition, many aspects of the Application Domain remain stable over a given period of time; however some aspects may drift or change radically. Legislation may change during the lifetime of a project’s development, or public attitude to risk in the project’s domain may change as the result of some accident that occurs during a project’s development.

6.4 The Use of Appropriate Techniques

The most useful representation of the Problem Context is a context diagram. There is a type of context diagram, such as that recommended by [5], which are used to record the boundary between the Application Domain and the Machine. The production of such diagrams is portrayed as a simple step where the Machine under construction is drawn in the centre of a piece of paper, and the other parties that should communicate with the Machine are drawn with connecting arrows to indicate information flow. The purpose of such a context diagram is to identify the parties that should communicate with the target Machine, their interaction with the target Machine, and the actual nature of the boundary between the Machine and its environment. Figure 9 is an example of a traditional context diagram, for a railway problem context.

![Diagram of Context Diagram]

Jackson observes [4] that this type of context diagram is quite limited in its descriptive capability in that it does not record any of the interactions between elements of the Application Domain (e.g. drivers and trains). Indeed, there are a number of tools that support such diagrams that actually preclude the inclusion of such information.
Often the interactions that occur within the Application Domain independently of the Machine are a great source of information for the Satisfaction Argument. Consider the construction of an Air Traffic Control system that must liaise with a number of other aircraft information systems, Figure 10.

![Figure 10 – Extended Air Traffic Control System Context Diagram](image)

In this particular example there were interactions occurring between the aircraft that were being controlled and Location System 1, and between the aircraft and Location System 2. These interactions were critical to understanding the information input to the Air Traffic Control System because the information from the two Location Systems could actually be in conflict. This conflict was only understood by looking beyond the traditional context diagram to see that Location System 1 received its information manually and Location System 2 received its information automatically (and hence Location System 2 was more reliable).

Typically we therefore use what we term Extended Context Diagrams, e.g. Figure 10. In an extended context diagram the interactions between all entities are shown, not just the interactions between the entities and the target system. It is interesting to note that Mullery’s method CORE [6], whilst not explicitly including these diagrams, does support the recording of these sorts of dependencies in its Tabular Collection Forms. However, structured analysis techniques such as [5] explicitly bar the recording of such information.

We have found that extended context diagrams are an extremely useful aid to capturing domain information and are also an effective elicitation aid. However they clearly only focus on active information and relationships (e.g. Air Traffic Controllers enter information into the Air Traffic Control system) and are not effective for capturing other types of information such as passive relationships, mathematical properties, etc. See Table 5 in section 8.4 for a variety of other approaches that can be useful.
7 Identifying Stakeholders and Eliciting Requirements

7.1 Introduction
The problem context provides a framework and vocabulary for discussing and describing a requirements problem, but it does not in itself provide the description of the information required by the Satisfaction Argument – that is the Domain Knowledge properties, the System Specification, or the Requirements themselves. For that, it is necessary to gather, or elicit, information from stakeholders. In this section, the main sources of requirements information are discussed and the classification of that information into the three types is described.

7.2 Process Steps
In order to identify the relevant stakeholders and elicit the requirements information, a number of process activities are required. These are listed in Table 2.

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Identify the relevant sources of information.</td>
</tr>
<tr>
<td>2</td>
<td>Where the number of possible stakeholders is too great, choose a smaller set that is representative.</td>
</tr>
<tr>
<td>3</td>
<td>Elicit information about the Application Domain</td>
</tr>
<tr>
<td>4</td>
<td>Elicit requirements from each stakeholder</td>
</tr>
<tr>
<td>5</td>
<td>Be prepared to re-visit this stage as new information is acquired.</td>
</tr>
</tbody>
</table>

Table 2 – Process Steps for Requirements Elicitation

At the completion of this process, we are ready to start analysing the information that has been elicited and developing the Specification (with a view to establishing the Completion Criteria identified in Section 5). In reality, the process of Requirements Elicitation and the process of Analysing and Writing Requirements (see Section 8) are undertaken iteratively.

7.3 Discussion
Requirements information is elicited from a variety of sources.

*Application Domain:* This will include both the physical environment (for example information about trains, weather, pollution, etc.) but also the social/political environment (for example market pressures, regulation, etc.).

*People:* This will include stakeholders who use the system, stakeholders who will pay for the system, or stakeholders who more generally will have an interest in the development of the system. Such people might include for
example stakeholders who will be directly affected by the introduction of the system or stakeholders who regulate this kind of system.

Documents: This will include documentation such as working practices, existing system documentation, interface definitions, etc. Documentation is nearly always the least reliable source of information.

The range of applicable stakeholders for any development is likely to be large. Indeed, the number may increase over the development of a large system resulting in a need to re-visit this stage. The number of stakeholders is likely to be too large to be considered in totality. There will be insufficient time and resource to collate information from all stakeholders and so it is necessary to choose representative subsets or to make use of surrogates. As an aid to choosing a representative subset it is useful to consider all relevant phases of development and also to consider a statistical sample of a larger collection.

The key tasks in undertaking elicitation are to identify what we need information about and for most elicitation exercises we find that the Satisfaction Argument provides us with a clear focus for this. However, some information will be recalled readily by the stakeholder and some will not. The dominant problem in elicitation is not that stakeholders do not want to provide information, more it is that they do not know what they know (see the expression problem in Section 3.1).

We use a simple model to classify the knowledge a stakeholder possesses into three categories:

- **Non-tacit knowledge**: This is knowledge that a stakeholder knows that they know. Recall may be defective but it is knowledge that a stakeholder is likely to offer reasonably readily.
- **Semi-tacit knowledge**: This is knowledge that a stakeholder will recognise as knowledge when the fact is pointed out to them but will not usually be volunteered. Often knowledge that is non-tacit to novices becomes semi-tacit to experts, so the more expert someone is, the harder it may be to get information from them.
- **Tacit knowledge**: This is knowledge that a stakeholder will never even think of as knowledge and cannot be reliably retrieved at all.

In typical stakeholders, knowledge about the Application Domain is frequently tacit or semi-tacit. It will not occur to users that the engineer does not know what an ‘arrival’ is in airport management, or that signal spacings are calculated according to train braking distances in railway signalling. Techniques to uncover these assumptions are required and observation-based approaches are particularly useful here.

### 7.4 The Use of Appropriate Techniques

Many of the skills for eliciting information from people are inter-personal in nature. Skills such as ‘active listening’, assorted observation techniques, the use of mixtures of open/leading/limiting/closed questions to focus an interview, and ‘structured
interviews’ are well discussed in the literature [7] and we have found them all to be useful. Some specific techniques for identifying particular types of information are identified in Table 3.

<table>
<thead>
<tr>
<th>Type of Technique (in order of increasing realism)</th>
<th>Means by which Information is Elicited</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>Generates simple lists of ideas, used as a baseline for further work</td>
<td>Brainstorms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Unstructured interviews</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Structured interviews</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Uses some form of dramatised artefact, typically includes more context and dimension to the problem</td>
<td>Storyboards</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Prototype mockups</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Observation</td>
</tr>
<tr>
<td>Animated</td>
<td>Uses stakeholders, typically involving some form of “acting out”</td>
<td>Role playing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scenario Analysis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Observation</td>
</tr>
<tr>
<td>Technological</td>
<td>Uses some form of working functional prototype</td>
<td>Operational Prototyping</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Observation</td>
</tr>
<tr>
<td>Trial Implementation</td>
<td>Involves stakeholders directly in the development</td>
<td>Rapid Prototyping</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RAD</td>
</tr>
</tbody>
</table>

Table 3 – Techniques for Requirements Elicitation

Techniques can broadly be defined as either ‘asking’ (good for semi- and non-tacit) and ‘observing’ (good for tacit). Observation-based techniques are one of the most effective ways of acquiring tacit knowledge. When observing, it is important to be aware that the implicit and social activities are just as useful as the formal technical activities that are being undertaken. It is vital for the analyst to be sure that they understand exactly the situation they are observing. Supplementary questioning is a perfectly good technique to be used with observation. Common observation pitfalls usually relate to the observer being too constrained by the current solution that is in place. It is important to look beyond the System that is currently in place so as to be confident as to what the actual problems are that are being solved.

The purpose of this stage is to identify all the relevant sources of information and to acquire from them information relevant to the actual problems. This information must be recorded and re-presented to the stakeholders for confirmation.
8 Analysing and Writing Requirements

8.1 Introduction

The purposes of this stage are to:

a) organise all the Application Domain, Requirements and Specification material in a coherent way;

b) derive, through analysis, and record, in appropriate notations, the System Specification;

c) demonstrate the correctness of the Specification, in terms of meeting the stakeholder needs given in the Requirements.

This stage is an iterative one with the previous stage. For example, it may become apparent that more elicitation is required, or additional stakeholders are needed. In particular, we must ensure that sufficient Application Domain properties are recorded to show that the Specification, when implemented, will allow the Requirements to hold.

REVEAL recommends the use of scenarios (the term ‘Use Case’ is sometimes used) to assist both in elicitation and writing. In particular, scenarios can provide an effective way of deriving specifications.

8.2 Process Steps

The information that is recorded is the information obtained through requirements elicitation and analysis. The process steps recommended are given in Table 4.

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Decide upon a large-scale structure of the requirements documentation (see Section 4.5)</td>
</tr>
<tr>
<td>2</td>
<td>Identify potential scenario types through initial discussions with stakeholders. Discussions provide a means of validation.</td>
</tr>
<tr>
<td>3</td>
<td>Record the scenarios, using the identified component types as a guide to recording information.</td>
</tr>
<tr>
<td>4</td>
<td>Use the scenarios to derive the necessary behaviour of the Machine.</td>
</tr>
<tr>
<td>5</td>
<td>Record all relevant facts and assumptions about the Application Domain.</td>
</tr>
<tr>
<td>6</td>
<td>Write down the required System Specification using appropriate notation.</td>
</tr>
<tr>
<td>7</td>
<td>In the course of doing the above ensure all terms are defined.</td>
</tr>
</tbody>
</table>

Table 4 – Process Steps for Analysis and Writing of Requirements
8.3 Discussion

This section discusses the use of scenarios in elicitation and analysis of requirements.

The role of a scenario is to provide a narrative description of some behaviour of (usually but not always) a stakeholder in carrying out some desired activity to achieve a goal. This goal should be part of the Requirements that need to be achieved.

The scenario provides a description that has an initial state, some specific behaviour, and a specific end-point. Where the stakeholder will make use of the Machine as part of the desired activity, the scenario usually describes how the Machine will participate with the stakeholder, and how the Application Domain behaves in cooperation with the Machine. This description can be used to define requirements and explain how a stakeholder would expect some particular activity to behave. Scenarios can be expressed in natural language, though more formal notations such as UML activity diagrams [8] can also be used. The advantage of a good scenario is that it should enable the stakeholder and analyst to arrive at a shared understanding, and to appreciate the role of the Machine and the World in achieving the requirements.

Consider the following example of a scenario, whose goal is that a customer pays a shop for the goods they want.

“Customer browses in shop and selects goods. They go to sales desk and decide how to pay. If the value is less than £5 they will pay by cash. Otherwise they will decide between debit and credit cards by checking balance in bank account. If there is enough, they use a debit card, otherwise they use a credit card provided credit limit is high enough.”

Scenarios are descriptions of behaviour within the World and it is this that makes them so useful for capturing requirements. They help stakeholders identify the relevant aspects of the World which need to be recorded to support the requirements statements.

Scenarios can be identified very early on during the RE process. This is because they can be described in relatively little detail and yet still provide valuable information. Early scenarios are sometimes termed “Operational Concepts”. As scenarios are captured, they can be organised in a variety of ways. They can be organised according to specific function types, devices or Application Domain concepts, or by viewpoint. The scenario information must be managed throughout the development to ensure that the scenarios still describe correct behaviour – particularly for long-term developments.

As noted earlier, notations do exist for recording scenarios, however the key issue in recording scenarios is not really in the choice of notation but more in an understanding of the component parts of a scenario. The following scenario components are recommended.

A description of the initial system state that effectively defines the applicability of this scenario.

A description of the flow of events that describe the normal flow of activity through the scenario.
A description of the exceptions that might occur within the scenario. That is, if the normal flow of events is disturbed, perhaps because of some error condition, what is the required exceptional behaviour?

A description of any other activities that might occur during the operation of the scenario.

A description of the final state of the domain at the conclusion of the scenario.

We identify two specific uses for scenarios once they have been captured.

- For establishing agreement between all relevant stakeholders and the RE analyst on the actual requirements that are to be implemented.
- For investigating how a Machine might be used by a stakeholder and therefore for deriving system specifications.

Scenarios often include a description of how a Machine will be used. It is through the identification of this description that Machine functions and constraints can be identified. Indeed, if the Machine functions are already known, then these themselves can be used as drivers for structuring the scenarios. More usually though the instances of use of a Machine must be derived from the scenario. This derivation may be straightforward, but often the declarations of Machine use are implicit rather than explicit and must be identified through discussion with the relevant stakeholder.

Consider the shopping example again.

“Customer browses in shop and selects goods. They go to sales desk and decide how to pay. If the value is less than £5 they will pay by cash. Otherwise they will decide between debit and credit cards by checking balance in bank account. If there is enough, they use a debit card, otherwise they use a credit card provided credit limit is high enough.”

The three highlighted phrases describe potential uses of some Machine (a point of sales system for example) that might be used to form the basis of a specification for that Machine. Of course, if some other Machine was being considered then different specifications might emerge.

We note that the capture of error behaviour within scenarios can be particularly problematic. There are typically many ways that a scenario can fail, and the description of all of the possible combinations during the capturing of scenarios can lead to an extremely large set of descriptions. Within REVEAL, we try to keep the number of error cases to a limited size during this stage. Instead we capture detailed error behaviour during the specification of the functional behaviour of the Machine.

We note that deciding on the number of scenarios required to completely describe the required behaviour is very hard. If too small a number is provided then a representative sample will not be achieved. If too many are written then there is a tendency to record every conceivable aspect of behaviour – this will be complete, but is unlikely to be cost-effective. Meyer [14] also notes that if the role of the scenarios is misunderstood invalid assumptions about the ordering of events can be made. This leads to systems that are unable to handle valid event sequences occurring in the domain.
8.4 The Use of Appropriate Techniques

This section describes notations used to document specifications.

There are many different kinds of requirement that must be recorded, and there are many different kinds of descriptive language (see [10] for a good summary). The many types of requirement information imply that no single notation is sufficient to capture the relevant types of information. An eclectic approach is therefore more likely to be successful. There are a number of different kinds of information that must typically be described – we can consider properties of the domain, properties of the requirements, and properties of the specifications. A variety of approaches can be useful for describing this information.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Relationship Diagrams</td>
<td>Useful for identifying both concrete and abstract entities and their relationships (see for example [5, 8]).</td>
</tr>
<tr>
<td>Entity Relationship Class</td>
<td>Diagrams</td>
</tr>
<tr>
<td>Domain-Specific Notations</td>
<td>For example electrical circuit diagrams, gas turbine acceleration graphs, etc.</td>
</tr>
<tr>
<td>Engineering Mathematics</td>
<td>For example Braking Formulae for trains.</td>
</tr>
<tr>
<td>Discrete Mathematics</td>
<td>For describing discrete properties and constraints (see for example [9]).</td>
</tr>
</tbody>
</table>

*Table 5 - Techniques for Description*

There are many different kinds of requirements – a candidate list is identified below, though there may of course be others for particular domains or applications (see [10] for alternative lists).

- Scope (requirements that define the extent of the Machine)
- Operational (includes requirements about Machine functions and performance)
- Interface (includes interfaces with users, other systems and the environment)
- Safety
- Security
- Integrity (includes Reliability, Maintainability and Availability)
- Design constraints (requirements on things in the Machine)
- Process (requirements on how the Machine is developed, e.g. cost and timescale)
- Standards

REVEAL includes guidance on the writing of requirements of each of the above types, but this paper only briefly considers the recording of specification functions.
(whose relationship with requirements are typically given via scenarios). The vocabulary of the function definition must include only terms meaningful in the application domain. Functions can be initiated by the World or by the Machine. If the former, they really contain two separate descriptions: the application-controlled stimulus and the Machine-controlled result of the function.

Functions can be described in English or in mathematics. In either case, we are interested in the net effect of the function, not in how it is achieved. Therefore it is usually a bad idea to specify functions procedurally. Unfortunately, the tradition of Structured Analysis (e.g. [5]), which has been carried over into most object-oriented methods (e.g. [8]), is to define functions in terms of dataflow diagrams, with the lowest level processes defined in pseudocode (if at all). This is not consistent with REVEAL as these lower level diagrams are sketches of designs, not statements of requirements.

Whatever notations are used for functional specifications, the following elements should be included.

- **Stimulus** – what causes the function to happen?
- **Information used** – inputs to the function, or knowledge about the World.
- **Information generated** – outputs to the World, or changes in state.
- **Precondition** – under what circumstances is the function valid?
- **Effect** – what does it actually do?

The precondition of a function is an obligation on the user (person or system) of the function. It is an assumption that the developer of the function can make in order to make their job easier. For example, it may be reasonable to assume that certain combinations of inputs can simply never happen. These preconditions must be derived from the properties of the domain. Never assume as a precondition what you would like to be true, only what you can prove to be true. For example, if a controller receives a request from an airliner for a flight level, it is not safe to assume that the plane is where it thinks it is; but it is reasonable to assume that it is below 100,000 feet because airliners can’t fly that high. The effect of a function should be described by defining the required relation between the values of the information generated and the information supplied. Additional information can be attached to a function description, for example volume, performance or integrity requirements. REVEAL has been used with a number of notations, including Fusion [12].
9 Verification and Validation

9.1 Introduction

The previous sections have primarily identified the activities involved in capturing requirements information. None of this information is of any use if it has not been confirmed as accurate. Verification is the activity of checking whether “we are building the thing right”; and validation is the activity of checking whether “we are building the right thing”.

9.2 Process Steps

This process stage is implemented when requirements information is available for inspection. The process steps for this stage are identified below.

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>During elicitation, confirm your understanding with the stakeholder as the facts are identified.</td>
</tr>
<tr>
<td>2</td>
<td>Undertake a formal review of your findings with stakeholders after the documentation has been produced for validation purposes. Where appropriate use prototyping, models, and simulations.</td>
</tr>
<tr>
<td>3</td>
<td>Decide upon the verification attributes that are relevant for this project.</td>
</tr>
<tr>
<td>4</td>
<td>Verify the verification attributes using automation where possible, but formal review otherwise.</td>
</tr>
</tbody>
</table>

Table 6 – Process Steps for Verification and Validation

Upon completion of verification and validation, assuming that any remaining conflicts have been resolved and the Completion Criteria are satisfied, the requirements information may be taken forward into use and development.

9.3 Discussion

9.3.1 Validation

At one level, the act of validation is something that must be undertaken continually during the requirements process. This undertaking applies both to the understanding of the Application Domain that is being learnt, and the understanding of the requirements for the solution. This validation occurs both at initial acquisition, and at the final stage of document production. Requirements elicitation is an imperfect process and misunderstandings are typically introduced.
9.3.2 Verification

There are many published lists of quality attributes but we use the following.

- Unambiguity – ensure terms are defined, no multiple meanings exist. The use of mathematics can greatly assist this attribute.

- Consistency – consider both syntactic and semantic consistency of any notations that are used. Many structured methods provide syntactic consistency checks, however the confidence that such checks provide is misleading. Real or semantic consistency typically requires logical reasoning. We must also cross-check the various requirement descriptions. For example, are there interface requirements for every external entity on the context diagram that directly interacts with the Machine?

- Completeness – the elicitation techniques described in Section 7, and the validation techniques described below, help to ensure all the requirements are captured. The Adequacy Completion Criterion confirms whether or not sufficient domain knowledge and specification statements are recorded to satisfy these requirements.

- Feasibility – is it actually possible to build a Machine that will meet these requirements? Note that consistency does not imply feasibility.

- Verifiability – it must be possible to demonstrate in a practical way that a requirement has been satisfied.

- Traceability – is the derivation and source of all information known? (This issue will be returned to in Section 11.)

- Usability – change is a major part of nearly all projects. To support activities such as impact assessment the requirements material should have a clear structure, appropriate cross-referencing, a minimum of duplicated information etc.

9.4 The Use of Appropriate Techniques

9.4.1 Validation

The following approaches are all useful guides for validating requirements information.

- Speaking the stakeholder’s language.

- Use prototyping – useful for agreeing upon user interfaces.

- Use simulation and modelling – similar to prototyping, but for other aspects (e.g. performance).

- Playback of consequences – the drawing out of consequences of particular pieces of information helps to ensure completeness.
Speaking the Stakeholder’s Language

The user and the analyst will think of the Machine in different terms – typically, the analyst will think more abstractly than the user. It is essential that the analyst does not just translate from the user’s explanation into their own language; they must also translate back into the user’s terms to validate their understanding.

Use Prototyping

Prototyping is particularly useful for assisting stakeholders to see what it is that they might get. Prototypes can range from simple paper storyboards to full system mock-ups. When using a prototype it is important to understand for what purpose the prototype is being developed. For example, the screen to be used by a radio dispatcher might be mocked up to see if it provides all the facilities that the dispatcher needs. This is then the purpose of the prototype, and it must be evaluated with the dispatcher for this specific reason – not for general aesthetics, but for the use of the dispatcher in this particular role. Users should be encouraged to criticise the prototype – otherwise the exercise will not present the information that is required. All too often, prototypes are confused with demonstrators – whose purpose is to convince users that they are getting the right thing, rather than finding out any problems. The questions asked about the prototype are as important as the prototype itself.

Whilst prototyping can be very beneficial, there are also some dangers in its use. If you show a user a screen with a particular layout of buttons, they may be constrained to think in terms of buttons for invoking the functions when in fact some completely different notion, like displaying things on a map and pointing at map features, would be better. If you show a user a screen and only ask about the colours, you may not find out that there is a whole function missing – the user may just have assumed that you weren’t interested in that at the time. Conversely, if you show a screen simply to get the colour scheme right, the user may be disappointed when the final product has a different layout – even though you didn’t intend the layout in the prototype to be significant. Of course, when you present a prototype the user will make unexpected comments and discuss incidental features. You will certainly learn something from these comments – so be receptive to them. However, do make clear whether the features commented on are really part of the purpose of the prototype.

Use Simulation and Modelling

Simulation and modelling are similar to prototyping, but are typically concerned with issues such as operational capability, performance, reliability, maintainability, availability, logistics, etc. They can be used to establish whether proposed Machines can actually meet the requirements, and also to explore alternatives. Simulations can be used for many different types of requirement, and each type has its own modelling discipline.

Operational models explore the behaviour of the proposed Machine in its environment. For example one might build a model of airspace and aircraft movements to explore the effects of proposed control regimes. Performance models are similar but may use numerical or analytical techniques to evaluate expected throughput and response times. Reliability, Maintainability and Availability (RMA)
models can be used to evaluate the expected availability of complex systems in the face of failures. Logistics models are used to evaluate requirements for spares holding, fitment programmes, maintenance regimes and similar issues.

**Playback of Consequences**

Validation by analysis can also be an effective medium. The analyst may be able to realise consequences of the specification that the users would not think of. These consequences should be explained to the stakeholders. They may be undesirable limitations. It is better to get the limitations out in the open as soon as possible. They can be negotiated at this stage; later is too late.

In one project we have worked on, the client stated some security requirements for the Machine. We wrote a formal specification of these requirements and used mathematical reasoning to prove certain properties that were implied by what the client had asked for. We then expressed these properties informally and explained them to the client. The properties were things like “you won’t be able to delete this record unless you are the person who last corrected it”. When they realised the consequences of what they had asked for, they changed their minds about the security requirements.

Finally, it is necessary to consider completeness as part of validation. One way of doing this is for the Requirements Engineer to explore what a stakeholder expects to happen and some techniques are well documented by [11]. For example, asking what the stakeholder is looking forward to when a Machine will become operational. This sort of exploration can often uncover tacit assumptions (and thus address the expression problem, see Section 3.1) that a stakeholder may have. Considering the impact of failure (for example security threat analysis or safety hazard analysis) can also be a valuable aid to ensuring completeness.

### 9.4.2 Verification

The more formal the approach that is used for capturing requirements, the more automated the verification checks identified in Section 9.3.2 can be. However, we find that formal peer review is still the most widely useful technique for verifying requirements material.

The Adequacy Completion Criterion (Section 5) is the key to assessing the completeness of Domain Knowledge and Specifications.
10 Conflict Management

10.1 Introduction

In Section 3.1 we argued that conflict is a natural part of the RE process and one of the most significant challenges of RE is how to manage conflict successfully.

Conflicts have traditionally been viewed as undesirable. In fact, most software engineering techniques and approaches are based upon resolving and removing conflicts as soon as is possible – or ensuring that conflicts are never introduced into the process. Inconsistent or conflict-ridden descriptions are deprecated because they do not allow formal reasoning, or betray a lack of understanding.

However, it is our view that conflicts are a fundamental part of engineering a complex system. They are fundamental because the system must meet the requirements of all the individual stakeholders and stakeholders will disagree. So to ignore the existence of conflict to us appears naïve and damaging to the overall process. Stakeholders will disagree because of fundamental conflicts that arise from differing technical, political, social or economic views. They will also disagree because of mistaken understandings or misplaced assumptions. A proactive support for conflict management is likely to provide a richer knowledge of the domain and the Machine, ensure that more informed design decisions are made, and increase the likelihood of constructing systems that work. Having recognised that conflicts might in fact prove useful, it is necessary to understand how to identify the different kinds of conflict.

10.2 Process Steps

The process of conflict management is one that can be applied throughout the whole development life-cycle.

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Identify conflicts within the set of Requirements and the set of Domain Knowledge.</td>
</tr>
<tr>
<td>2</td>
<td>Classify the conflicts according to the categories defined in Section 10.3.</td>
</tr>
<tr>
<td>3</td>
<td>Resolve the conflicts through a process of negotiation.</td>
</tr>
<tr>
<td>4</td>
<td>Maintain and record the conflict management information throughout the life-cycle.</td>
</tr>
</tbody>
</table>

Table 7 – Process Steps for Conflict Resolution

10.3 Discussion

A distinction in type of conflict that we have found useful is to separate fundamental conflicts from superficial conflicts. Superficial conflicts are usually disagreements
about design decisions. For example a conflict in constructing a railway signalling control system user interface from a large mimic display or a VDU can be resolved by considering what the underlying requirement is – who requires the data and what is that data to be used for. Stepping back to the underlying requirements can often resolve these superficial conflicts, which tend to emerge through the specification, S.

A typically more challenging type of requirement conflict to solve is that which emerges through a conflict in the underlying requirement itself – e.g. a wish to automate a process plant and a wish to maintain or increase the size of a workforce. These fundamental conflicts typically emerge in \$ and may not be resolvable by agreement – in this case, a political decision on what is to be done may have to be taken by whoever is in charge.

An additional categorisation of conflict that can be made is to consider intrinsic and extrinsic conflicts. Intrinsic conflicts are logical contradictions, for example a need for an airport to increase the number of landings per hour to satisfy the airlines, but also to decrease the number of landings per hour to satisfy the local residents. Extrinsic conflicts are not in themselves contradictory, it is just that it is not feasible (currently) to construct a Machine that will satisfy both of them – e.g. construct an aeroplane that can fly both faster and more quietly. As opposed to intrinsic conflicts, that will never be reconciled, extrinsic conflicts might be removed by future developments (for example the modern bypass engine has to an extent addressed the example extrinsic conflict). In addition, not all extrinsic conflicts will be apparent during the RE phase – some will only emerge as development proceeds (software development processes that do not support analysis of timing behaviour are a good example of emergent conflicts where it is only as the working system is built that the conflict between functionality and performance emerges).

All possible combinations of conflicts can occur, and whilst the divisions are not completely independent, a cross-reference of superficial/fundamental against intrinsic/extrinsic does provide a useful taxonomy.

<table>
<thead>
<tr>
<th></th>
<th>Intrinsic</th>
<th>Extrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fundamental</td>
<td>Increase traffic</td>
<td>Increase speed</td>
</tr>
<tr>
<td></td>
<td>Decrease traffic</td>
<td>Decrease noise</td>
</tr>
<tr>
<td>Superficial</td>
<td>Use flight strips</td>
<td>Use touch sensors</td>
</tr>
<tr>
<td></td>
<td>Use data blocks</td>
<td>Use high resolution</td>
</tr>
</tbody>
</table>

Table 8 – Examples of Conflict Types

An important class of fundamental extrinsic conflicts arises between requirements about functionality, cost and timescales. Trade-offs are often needed between such requirements for the combined set to be feasible. Conflict management therefore plays a crucial role in Value Engineering (see Section 3.3).
10.4 The Use of Appropriate Techniques

Understanding the types of conflict that can occur is clearly a step towards successfully managing conflict, but it is also necessary to understand how best to resolve that conflict once it has been identified. Within REVEAL we recommend three outcomes of conflict resolution [17].

- Total elimination of conflict
- Dynamic resolution of conflict
- Prioritisation of conflict

Total elimination requires that whatever the state of the system is, the conflict cannot occur. Total elimination requires one party to change their mind and agreement enables the conflict to be resolved (e.g. it is agreed to increase traffic rather than decrease it). Total elimination of a conflict cannot always be achieved, but if it can be done it will result in a change to $\mathcal{R}$, or occasionally a change to $\mathcal{D}$.

Dynamic resolution requires a change in a specification such that the requirements in conflict can in turn hold at different points in time. For example, it might be that during daylight hours, the number of flights to an airport will increase, but during night-time the flights will be dramatically reduced.

Conflict resolution through priority requires that requirement weight can be distinguished – i.e. it is possible to determine that some requirements are more important than others. Elaborate priority levels can be used, but often a simple recognition that some requirements are mandatory and some are optional can prove useful.

Whatever the approach to managing and resolving conflicts, successful negotiation is always required. It is important to avoid ‘win/lose’ situations, Boehm’s work [13] on WIN/WIN negotiation is useful here. The approach provides guidance on the identification of common ground between conflicting parties, such that all parties can proceed to a resolution where each is able to take something from the conflict. (We note that the using and maintaining Viewpoints can be particularly useful in managing conflicts as it provides the means by which identified conflicts can be tracked). Once identified, conflicts need not necessarily be resolved immediately as later resolution can sometimes be carried out under more clear and accurate circumstance – i.e. more information is known, making the resolution more informed. However, if conflict resolution can be carried out quickly, it should be done so. If conflict resolution cannot be carried out quickly, then it is necessary to attempt to identify areas of common ground between conflicting parties that will at least enable progress to be made in identifying possible points of compromise.

One of the important aspects of conflict management is that the reasons why requirements cannot be satisfied should be identified. Identifying the reasons for the conflict helps to understand the implications of choosing to resolve a conflict through the different mechanisms available. Successful conflict management will then highlight and increase the understanding of potential areas of misunderstanding, and also provide a focus for further areas of elicitation and analysis. Methods that do not
support a proactive approach to conflict management are likely to result in unsuccessful projects.

Finally, we note that having identified and resolved a conflict, the information that was used in resolving the conflict should be maintained since, over the lifetime of a project, conflicts may not stay resolved. Conflict management is something that must be undertaken across the whole requirements engineering life-cycle, and so the identified process steps need to be applied throughout the REVEAL process.
11 Maintenance and Management

11.1 Introduction

Requirements information is necessary from the initial elicitation exercise right through to operation and maintenance. As the life-cycle progresses, requirements grow and change as issues are clarified, contracts change, and understanding increases. Requirements management is concerned with supporting and controlling requirements information throughout the life-cycle - ensuring that the requirements integrity, and thus the integrity of the system information, is maintained. For example, when change occurs mechanisms must be in place to allow the change to be successfully accommodated, and the implications of the change addressed.

The critical barrier to effective requirements management is scale. The more requirements you have, the harder it is to know what you’ve got, what the interdependencies are, and what the implications are of any particular change. Scale can affect requirements management in a number of ways. Firstly the amount of information that must be recorded may be large. Secondly, the number of stakeholders who may be affected by any change may be large. Thirdly, the duration of a project may be large. Successful requirements management requires mechanisms that will help to control information regardless of scale.

11.2 Process Steps

There are really two sides of the process for maintenance and management. There are the process steps that must be followed at the start of a project (steps 1-4 below), and the process steps that must be followed during the project (steps 5-6 below).

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Define the traceability approach.</td>
</tr>
<tr>
<td>2</td>
<td>Define the configuration management process.</td>
</tr>
<tr>
<td>3</td>
<td>Choose tool support for traceability and configuration management.</td>
</tr>
<tr>
<td>4</td>
<td>Define and agree upon a change management process.</td>
</tr>
<tr>
<td>5</td>
<td>Ensure that traceability relationships are recorded during the development.</td>
</tr>
<tr>
<td>6</td>
<td>Follow the change management process for all changes.</td>
</tr>
</tbody>
</table>

Table 9 – Process for Maintenance and Management
11.3 Discussion

Requirements information is typically contained within documents. Even when tool support is used, the concept of a ‘document’ (even if it may be a virtual one containing information from a number of tools) is still a useful one. In order to manage these documents, it is necessary to think about the management of the documents themselves, and the management of the information that is contained within them.

As a life-cycle progresses, it is useful to be able to take a snapshot of the current state of the requirements. Any set of requirements for a system is referred to as a baseline. Managing baselines can be performed effectively using conventional configuration management processes. Such processes provide the means to know what is current, know what was previously current, and know what changes have been made.

In order to be able to identify a baseline, each requirement must be identifiable. Identification can often be achieved through labels. Labels can take many forms, and will be dependent upon the method used to express the requirements or the tools that are used to support requirements engineering. Labels should aid the identification of the baseline, and, if required, its significance. Baselines can have control numbering schemes or be labelled in relation to the milestone to which they relate.

In addition to managing the requirements documents, it is also necessary to understand the links between the requirements and other components so that change management can be successfully managed. This understanding is provided by traceability.

Traceability is a mechanism for recording relationships between different components. It provides a means of saying, for example, that requirement 1.1 is implemented by design unit 3.4.5; and that, for example, design unit 3.4.5 implements requirements 1.1, and 1.4. In this way, traceability provides the mechanism for checking that all requirements have been implemented, and that all that has been implemented was required.

With these links in place, it becomes possible to understand and assess the impacts of any change. In the above example, for instance, a change to design unit 3.4.5 would necessitate a re-examination of requirements 1.1 and 1.4 to check that they were still correctly implemented.

11.4 The Use of Appropriate Techniques

It is necessary to understand the extent to which a change affects a system. The components included within the traceability model will therefore depend on the system in question, but could include hardware, software, people, documentation, etc. Traceability can function in a number of dimensions, and in constructing a traceability framework, it is necessary to determine which kinds of relationships are appropriate to record.
Downwards and upwards traceability is concerned with looking down, and back up, the life-cycle - hence it is sometimes referred to as vertical traceability. Vertical traceability provides the means for associating a requirement with design components further on in the life-cycle, and with design components back up the life-cycle. Vertical traceability makes it visible where requirements are implemented, and where design components have come from. Vertical traceability therefore typically deals with information of different life-cycle types. Traceability in the horizontal dimension (sideways traceability) is concerned with associating information with other information at one particular documentation level. For instance one design component is associated with another, or one requirement with another.

The majority of traceability links are typically many-to-many. That is individual requirements will be satisfied by many design elements, and individual design elements will satisfy many requirements. However other relationship cardinalities are possible. One-to-one relationships are perhaps the rarest - i.e. a single requirement is met by a single design element that does nothing else. Whilst of course this is possible, it perhaps suggests a requirements decomposition that is not as independent of the design process as it might be. One-to-ones do exist for certain kinds of development, but not many.

Traceability is simply a relationship between information. The information can be requirements, design features, design modules, or even code, and the relationship links the information together. The vertical relationship is usually a “derives” or “is implemented by” relationship. Traceability in its usual form allows a developer to trace the “life-cycle” of a requirement through system development. It is therefore important to ensure that the source of the requirements is recorded and that each requirement can be traced back to its source. In many cases this will be a person or a minuted statement.

If a specification statement is not itself a requirement, it is traced to any requirement which it is intended to satisfy. In addition, any domain knowledge that is necessary to show that it satisfies the requirement is also traced to the requirement. The diagram in Figure 12 shows an example. It shows two kinds of tracing: solid lines showing...
that a specification is there to satisfy a requirement; dashed lines showing that domain knowledge is required to demonstrate that the specification meets the requirement.

Figure 12 – An Example of Traceability Relationships

The tracing suggests, but does not of course guarantee, that these relationships hold. It is possible to add satisfaction arguments to the tracing information. A satisfaction argument provides some justification of why these relationships are correct. Figure 13 shows the same tracing with a satisfaction argument added. Here there are two kinds of traceability link:

1. A **satisfies** link connects the satisfaction argument to the requirement which is being satisfied.

2. A **contributes** link connects the domain knowledge and specifications to the satisfaction argument that they contribute to.

This structure is called Rich Traceability².

² The term “Rich Traceability” was invented by Jeremy Dick of Telelogic DOORS
Figure 13 – A Simple Satisfaction Argument

The example here is very simple. In a real situation the correctness of the satisfaction argument may be far from obvious. Often it is necessary to carry out modelling or analysis to show that the argument is correct. For example the satisfaction argument about crossing the road might be supported by a scenario showing the relevant actions of the World and the Machine, as shown in Figure 14. The purpose of the scenario is to allow us to identify whether all the relevant aspects of Machine and World behaviour are captured.

Figure 14 – Supporting the Satisfaction Argument
Different kinds of model may be used to support satisfaction arguments. For example a requirement for trains to travel from London to Crewe in two hours might have a satisfaction argument supported by detailed performance models of the railway.

The only way requirements management can be successfully implemented is for certain base mechanisms to be in place. If the mechanisms are in place, then they provide a firm grounding for successful management activities, even when the scale of the problem suggests that such activity would be difficult to perform. One of the most frequent requirements management activities is the introduction of change. For the introduction of change to be successful, the traceability mechanisms identified must be in place, and then an appropriate process followed.

The process of change control can be thought of as a series of steps. The sequence of steps may not be entirely sequential, but are as follows.

- **Request** – someone requests a change to be made.
- **Consultation** – the change is discussed with all relevant stakeholders.
- **Evaluation** – impact analysis establishes the consequences of the change.
- **Approval** – if the change is worthwhile it is approved.
- **Documentation** – the scope and implications of the change are documented.
- **Implementation** – the change is implemented.
- **Tracking** – the progress of the change is traced, to ensure that it is completed.

When the introduction of a change introduces a conflict, it is necessary to resolve this conflict. Conflict management is discussed in detail in Section 10, but there are certain resolution strategies that are often used within change management. These include requirements trading, rejection of a proposed change, modification of the new requirement, modification of existing requirements, or discarding of the conflicting requirements.

Finally, we note that there exist a number of tools available for supporting requirements management. Requirements management is very amenable to tool support as tasks such as traceability, impact analysis and configuration management can all be automated to some degree.

The critical success factor in using a tool is the successful match of the process encoded within the tool with the processes encoded within the user organisation. Both processes need to be understood and consistent with each other.

REVEAL can be used with a variety of tools. For example, it has been successfully used with both Telelogic’s information and requirements management system DOORS® [15] and 3SL’s systems engineering toolset Cradle® [16].
12 Conclusions

Praxis Critical Systems has used REVEAL on a number of projects in the finance, aerospace, and rail industries. The method has been used with a variety of formal and informal notations and tools.

REVEAL addresses three key problems in Requirements Engineering:

- *It is difficult to get users to articulate what they want.*
  
  REVEAL provides a systematic elicitation process which focuses on the needs of all the stakeholders and on important properties of the World. It provides a disciplined way of deriving the specification of a Machine which will satisfy the users’ needs.

- *Requirements, once expressed, may be in conflict.*
  
  REVEAL acknowledges that conflicts exist in requirements and must be resolved to obtain a conflict-free specification in order for system development to proceed. REVEAL provides a process for recognising, classifying and resolving conflicts.

- *Requirements change over time.*
  
  REVEAL recognises that during the system lifecycle many factors, such as technological advances, will have an impact on the original requirements. The REVEAL method incorporates a change management process and actively identifies assumptions and requirements that are likely to change. Traceability is used to facilitate impact analysis and control when change does occur.

In summary, the REVEAL approach to Requirements Engineering described in this paper is an industrial strength engineering method. REVEAL has taken the scientific principles proposed by Michael Jackson and applied them in the light of our practical experience to produce a process and guidelines on implementation of that process.

Praxis Critical Systems continues to develop the REVEAL method and routinely uses it on its projects.
Document Control and References

Praxis Critical Systems Limited, 20 Manvers Street, Bath BA1 1PX, UK.
Copyright © Praxis Critical Systems Limited 2000, 2001. All rights reserved.

Changes history

Issue 1.0 (27th June 2000): First definitive issue for public distribution.
Issue 1.2 (24th July 2001): Added material on rich traceability.

Changes forecast

Process Illustrations will be added to sections 6-11. This paper will be maintained as REVEAL itself is developed.

Acknowledgements

REVEAL has been developed by many members of Praxis Critical Systems staff. We are grateful for the comments of Michael Jackson on our training course material and the subsequent changes to previous drafts of this paper. We also acknowledge the contribution that our clients have made to these ideas, both through working with us on projects, and also through the REVEAL training courses that we regularly run.

Document references


15  Telelogic UK Ltd, Chancery House 8 Edward Street, Birmingham, B1 2RX, UK. (www.telelogic.com)

16  Structured Software Systems Limited (3SL), Craven House, Michaelson Road, Barrow in Furness, Cumbria LA14 2RJ, UK. (www.threesl.com)